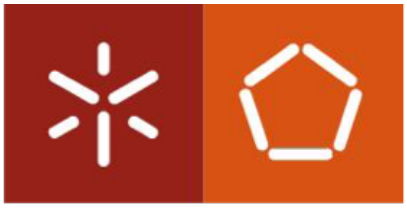

Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Diagnóstico de um Projeto Ágil

Gestão de Projeto de Software

Diana Lopes, nº pg38925

Conteúdo

1 Introdução **4**

2 Filosofia Ágil **5**

2.1 Desenvolvimento Incremental e Iterativo 6
2.2 Desenvolvimento Adaptativo 6
2.3 *Temeboxing* 7
2.4 Comunicação Presencial 7

3 Métodos Agile **8**

3.1 *Adaptive Software Development* 8
3.1.1 Conceitos 8
3.1.2 Fases do ASD 9
3.2 *Crystal Methods* 10
3.2.1 Métricas 10
3.2.2 Fases 11
3.3 *Dynamic Systems Development Method (DSDM)* 12
3.3.1 Surgimento do conceito 12
3.3.2 Conceitos 13
3.3.3 Fases do DSDM 13
3.4 *Extreme Programming (XP)* 14
3.4.1 Surgimento do conceito 14
3.4.2 Valores 15
3.4.3 Variàveis de Controlo 16
3.4.4 Papeis Adotados 16
3.4.5 Práticas Adotadas 17
3.5 *Scrum* 17
3.5.1 Surgimento do conceito 17
3.5.2 Vocabulário Adotado 18
3.5.3 Regras 18
3.5.4 Processo 19
3.6 *Test Driven Development* 20
3.6.1 Fases 20
3.6.2 Limitações 22

3.7 *Kanban* 22
3.7.1 Surgimento do conceito 23
3.7.2 Vocabulário Adotado 23
3.7.3 Processo 23

4 Diferenças das metodologias **25**
4.1 Necessidades 25
4.2 Categorias dos métodos 26
4.2.1 Definição das Categorias 26
4.2.2 Finalidade das Categorias 26
4.3 Comparação ente os métodos 26
4.4 Necessidade do uso de diferentes métodos 27

5 Diagnostico Ágil **28**
5.1 Foco do trabalho 29

6 Aplicação do Diagnóstico **31**
6.1 Projeto Académico 31
6.2 Projeto Empresarial 1 33
6.3 Projeto Empresarial 2 34

7 Conclusão **37**

8 Trabalho Futuro **38**

Capítulo 1

Introdução

O ensaio desenvolvido, realizado no âmbito da unidade curricular de Gestão de Processos de *Software*, tem como finalidade desenvolver uma metodologia de diagnóstico de um projeto Ágil.

Existem diferentes métodos Ágeis que são vistos, na sua generalidade, como uma boa técnica de evitar os métodos mais tradicionais, que, por vezes, não são os mais indicados para aplicar no desenvolvimento de um projeto. Hoje em dia, são poucas as entidades desenvolvedoras tecnicamente e psicologicamente capazes de implementar a 100% uma abordagem Ágil de uma forma rápida e eficaz.

Inicialmente é explicada a filosofia Ágil, como foram identificadas diferentes abordagens Ágeis em contextos do dia-a-dia, onde são explicadas as diferentes filosofias de cada uma, assim como uma breve explicação do seu surgimento.

Posteriormente, considerando o conhecimento adquirido, agrupa-se e compara-se os diferentes métodos Ágeis descritos, tendo em conta as suas principais características. Assim é percetível os diferentes pontos fortes e pontos fracos das diferentes abordagens.

Por fim, é elaborado um diagnóstico de um projeto Ágil no formato de um questionário. Este documento tem como finalidade ajudar a perceber, numa fase inicial do projeto, qual a abordagem mais adequada a seguir num determinado projeto. Para além disto aplicou-se este questionário em diferentes projetos: académicos e em contextos reais.

Capítulo 2

Filosofia Ágil

Na sua origem, os métodos ágeis eram denominados de métodos leves. Posteriormente, em 2001, os membros da comunidade reuniram-se em *Snowbird* e adotaram o nome Métodos Ágil tendo publicado o "Manifesto Ágil", documento este que reúne as práticas e princípios da metodologia de desenvolvimento [8]. Neste manifesto são de salientar os quatro valores do desenvolvimento ágil, definidos através da experiência em desenvolvimento de *software* e ajudando outros a desenvolver, sendo estes:

- Os indivíduos e as suas interações encontram-se acima de procedimentos e ferramentas;
- O funcionamento de *software* encontram-se acima da documentação abrangente;
- A colaboração com o cliente encontra-se acima da negociação e contrato definidos;
- A capacidade de resposta a mudanças encontra-se acima de um plano pré-estabelecido.

A definição destes valores não se trata de um desprezo aos elementos e ferramentas tradicionais focadas no desenvolvimento de *software*, mas sim do estabelecimento de uma escala de valores salientando a flexibilidade e colaboração em vez da rigidez de processos e planeamento clássicos.
Para além dos valores estabelecidos também foram definidos 12 princípios do desenvolvimento:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de *softwar* e de valor;
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas;
3. *Software* funcional é entregue frequentemente;
4. Os projetos devem ser criados em torno de indivíduos motivados. Dê-lhes o ambiente e o apoio que necessitam, e confie-os para começar o trabalho feito;
5. Os projetos devem ser criados em torno de indivíduos motivados. Dê-lhes o ambiente e o apoio que necessitam, e confie-os para começar o trabalho feito;
6. O método mais eficiente e eficaz de transmitir informações para e dentro de uma equipa de desenvolvimento é a conversa cara-a-cara;

7. *Software* funcional é a medida primária de progresso;
8. Os processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, programadores e utilizadores devem ser capazes de manter um ritmo constante indefinidamente;
9. A atenção contínua à excelência técnica e ao bom *design* aumenta a agilidade;
10. Simplicidade – a arte de maximizar a quantidade de trabalho não feito – é essencial;
11. As melhores arquiteturas, requisitos e projetos emergem de equipas auto-organizadas;
12. Em intervalos regulares, a equipa reflete sobre como tornar-se mais eficaz, em seguida, ajusta o seu comportamento em conformidade.

Alguns anos mais tarde foi formada a *"Agile Alliance"*, uma organização não lucrativa que promove o desenvolvimento Ágil. A metodologia Ágil não se trata apenas de um conjunto de regras e procedimentos, trata-se também de uma forma de pensar quando se está a trabalhar num projeto. Esta metodologia assenta num conjunto de valores e princípios, guiando assim programador nas diferentes fases do produto. Entre esses princípios é de destacar o desenvolvimento incremental e iterativo, desenvolvimento adaptativo a contrariar o normal perspetivo, *timeboxing* e o facto da comunicação ser preferencialmente presencial em tempo real. Princípios estes que serão explicados nas secções seguintes.

2.1 Desenvolvimento Incremental e Iterativo

O desenvolvimento incremental e iterativo trata-se por desenvolver um produto através de ciclos encadeados e repetidos [6], trabalhando em poucas tarefas de cada vez, figura 2.1. Este é um fator importante no desenvolvimento de um produto pelo facto de o cliente e a equipa melhoram iterativamente o produto até que este esteja completo. Em cada iteração são feitas as devidas alterações às diferentes componentes, acabando por ser modificações que ficam muito mais baratas pelo facto de serem passos pequenos.
Em métodos tradicionais de desenvolvimento é definido que o esforço significativo na fase de planeamento ajuda a evitar problemas e más decisões, porém o que se tem vindo a concluir é que estes métodos são pouco eficientes pelo facto de haver um grande grau de incerteza no contexto interno e externo do projeto. As alterações feitas nos requisitos, prioridades, tecnologias e possíveis soluções não só implicam alterações ao trabalho futuro assim como reformulam o trabalho realizado, levando assim a custos mais elevados por casa alteração realizada no produto.

2.2 Desenvolvimento Adaptativo

Ser uma metodologia adaptativa indica que há uma preocupação em reduzir o esforço no planeamento mantendo apenas um visão de alto nível a longo prazo, isto deve-se ao facto de quanto maior for o tempo de planeamento, mais vago é o planeamento desse período. Um outro fator é permitir ao cliente alterar e validar o produto ao longo do desenvolvimento do mesmo, são criados momentos onde o produto, plano, estimativas e abordagens passam a ser avaliadas e melhoradas[5].

Figura 2.1: Desenvolvimento Incremental e Iterativo.

2.3 *Temeboxing*

O *Timeboxing* é uma técnica usada para o planeamento do projeto. Na generalidade dos métodos de desenvolvimento ágil o trabalho é repartido em pequenos incrementos, minimizando assim a quantidade de planeamento necessário[9].
Cada iteração é um curto período de tempo denominado de *timeboxes*. A cada *timebox* são selecionados os requisitos a serem implementados, figura 2.2, sendo a restrição temporal a predominante desta metodologia.

Figura 2.2: Desenvolvimento usando uma Timebox.

2.4 Comunicação Presencial

Os métodos ágeis favorecem sempre a comunicação presencial de forma a manter uma ligação próxima entre a equipa, cliente e *stakeholders*. A comunicação verbal, ao contrário da escrita, facilita a troca de ideias, ajudando também às boas relações entre as diferentes partes envolvidas no projeto, criando também sentido de propriedade comum sobre o projeto.

Capítulo 3

Métodos Agile

O desenvolvimento agile tem como métodos principais[6]:

- *Scrum*;
- *Crystal Clear*;
- *Extreme Programming (XP)*;
- *Adaptative Software Development*;
- *Feature Driven Development*;
- *Dynamic Systems Development Method*;
- *Kanban*.

Métodos estes que serão explicados em seguida.

3.1 *Adaptive Software Development*

O *Adaptive Software Development* é uma proposta desenvolvida por *Jim Highsmith* em 2000 que foi criada especificamente para o desenvolvimento de Sistemas de Software complexos. Este método baseia-se na colaboração humana e na auto-organização das entidades envolvidas no projeto, surgindo quando estas cooperam de modo a criar resultados que seriam impossíveis de realizar se o trabalho fosse realizado individualmente.

3.1.1 Conceitos

Como todas as metodologias ágeis, o ASD baseia-se em alguns conceitos, sendo de destacar os seguintes:

- ***Component-Based***: Desenvolvimento de *software* em pequenas partes.
- ***Change-Tolerant***: Como visto anteriormente as mudanças no desenvolvimento de projetos são frequentes. É sempre favorável ter esta tolerância e ser algo sempre pronto a adaptar.
- ***Iterative***: Desenvolver o projeto em pequenos ciclos (iterações), de modo a que a implementação seja satisfatória para cada missão definida.
- ***Mission Driven***: Cada iteração do ciclo de desenvolvimento justifica-se através de uma tarefa que pode ser alterada ao longo do projeto.

- ***Risk-Driven***: Os pontos considerados, pela equipa de desenvolvimento, características de alto risco são priorizados.
- ***Time Boxed***: Uso de prazos tangíveis com a finalidade de forçar a equipa a definir, de forma não ambígua, as diferentes componentes do projeto logo desde inicio.

3.1.2 Fases do ASD

O método ASD tem três fases: Especulação, Colaboração e Aprendizagem [12], cada uma com a sua finalidade, como pode ser observado na figura 3.1.

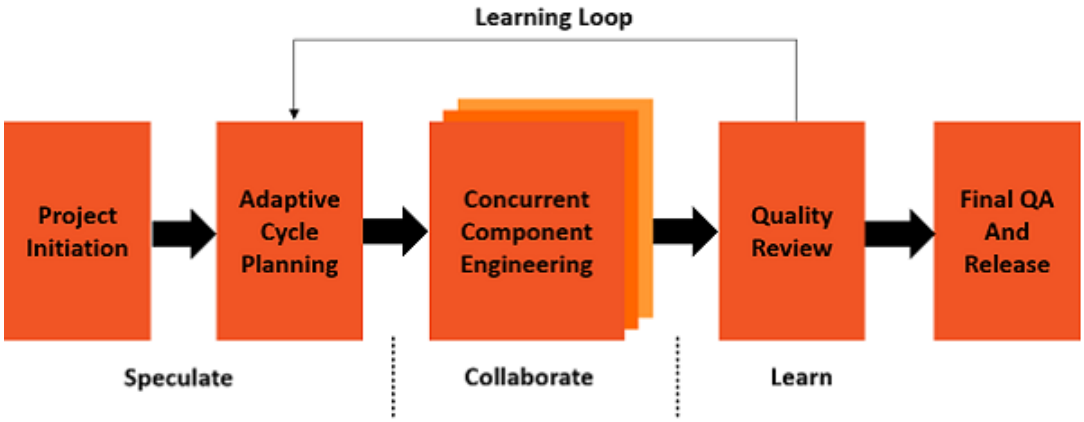

Figura 3.1: Metodologia ASD.

Especulação (Fase 1)

O termo planeamento é algo muito determinístico que determina, na sua generalidade, um grau de certeza sobre o resultado desejado. Nem sempre é o caminho idealizado para um determinado produto, impedindo, por vezes, a equipa de levar o projeto por caminhos distintos e inovadores em relação ao projeto inicial.

Neste método o termo planeamento é substituído pelo termo especulação. Ao especular a equipa não abandona o problema apenas reconhece a realidade da incerteza em problemas com uma resolução complexa. Esta atividade acaba também por incentivar a exploração e a experimentação do *software* em ciclos iterativos mais curtos.

Colaboração (Fase 2)

Um dos principais princípios do ASD é o facto das aplicações irem evoluindo ao longo do tempo. As aplicações complexas exigem colecionar um grande volume de dados sendo necessário a colaboração intrínseca entre toda a equipa.

Sendo necessário a que nesta fase haja uma gestão de projeto com técnicas tradicionais e criar ou manter o ambiente de colaboração.

Aprendizagem (Fase 3)

Por fim, a fase de aprendizagem do ciclo é muito importante para o sucesso do projeto. A equipa envolvida necessita de aprimorar o seu conhecimento de forma

contínua, usando algumas práticas já conhecidas, como:

- Revisões Técnicas;
- Grupos de foco com o Cliente;
- Retrospectivas do projeto.

As diferentes iterações necessitam ser curtas com a finalidade de que toda a equipa possa aprender com pequenos erros em vez de grandes.

3.2 *Crystal Methods*

Este método foi desenvolvido por *Alistair Cockburn* em 1990[18] com o objetivo de conseguir uma abordagem de desenvolvimento de *software* que premiasse o facilidade de realizar alterações durante o que *Cockburn* designava de "jogo cooperativo de invenções e comunicação de recursos limitados"onde o principal objetivo era desenvolver produtos úteis e funcionais, e com objetivo secundário preparar o próximo "jogo", isto é, a próxima etapa do projeto.
Os *Crystal Methos* são descritos como uma família de metodologias de desenvolvimento de *software* e, em similaridade com os cristais, possuem diferentes cores e rigidez. Estes diferentes estados (cores e rigidez) referem-se ao tamanho e ao nível crítico do projeto. Posto isto, é valido concluir que os *Crystal Methos* são focados nos talentos e nas habilidades de cada pessoa, permitindo que o processo de desenvolvimento seja moldado conforme as características da equipa de desenvolvimento. Misturam também, no desenvolvimento do projeto, a sua cultura com a proposta de desenvolvimento Ágil.

3.2.1 Métricas

Este método está assente em duas métricas. Estas métricas estão envolvidas na adequação do projeto, sendo elas:

- Número de pessoas envolvidas;
- Nível crítico do projeto.

De acordo com o número pessoas envolvidas na equipa cada uma destas métricas é caracterizada por uma cor[18]. Podendo variar nas seguintes cores:

- ***Crystal Clear***: metodologia leve, é constituída por equipas de uma a oito pessoas, podendo até chegar a doze elementos, em casos especiais;
- ***Yellow***: equipas de desenvolvimento de dez a vinte elementos;
- ***Orange, Orange Web***: equipas constituídas por vinte a cinquenta participantes;
- ***Red***: Equipas que envolvem cinquenta a cem membros.

Cada método, explícitos anteriormente, tem um grau de gestão e comunicação ajustado de acordo ao tamanho da equipa. Para além das cores são também utilizadas algumas letras para representar potenciais perdas devido à ocorrência de uma falha no sistema de desenvolvimento *software*, figura 3.2. Os diferentes níveis são caracterizados da seguinte forma:

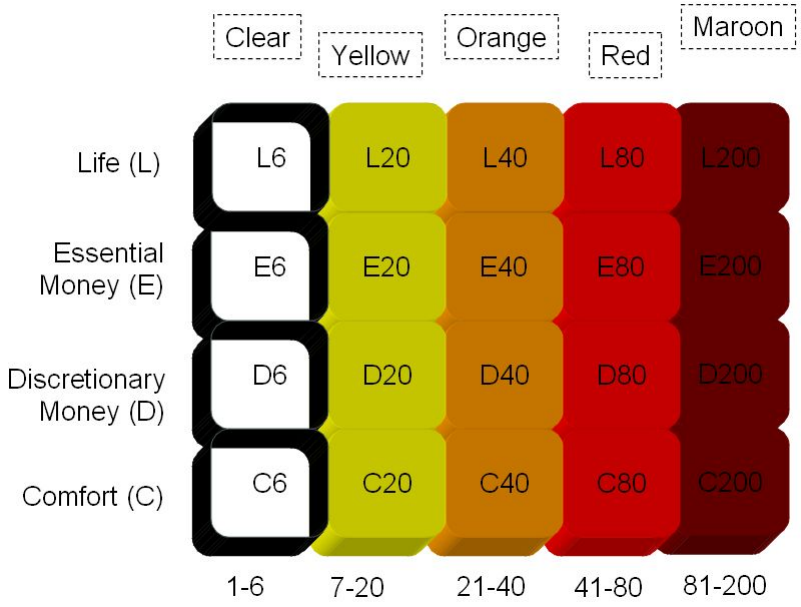

Figura 3.2: Parâmetros do nível critico.

- **C de *Confort***: casos onde a falha do sistema gera a perda de credibilidade do utilizador devido ao facto de não atender a este conforto;
- **D de *Discretionary Money***: casos onde a falha ocasiona a perdas de dinheiro, mas de valor inexpressivo;
- **E de *Essencial Money***: casos onde a falha do sistema leva a perda de uma quantias grandes de valores.
- **L de *Life***: casos onde a falha ocasiona a perda de vidas, tendo como referencia um *software* de piloto automático onde uma falha pode originar estragos enormes, como por exemplo a queda de um avião.

3.2.2 Fases

Esta família de métodos tem um ciclo de vida baseado em seis etapas, figura 3.3[17]. Sendo elas:

- **Staging**: Planeamento da próxiam iteração do sistema. A equipa seleciona os requisitos com maior prioridade a serem implementados na iteração e o prazo da sua entrega;
- **Construction Demonstration Review**: Construção, demonstração e revisão dos objetivos da iteração;
- **Monitoring of progress**: O processo é monitorizado em relação ao progresso e à estabilidade da equipa;
- **Paralelism and flux**: Aplicado no *Crystal Orange* onde as diferentes equipas podem operar com máximo paralelismo;

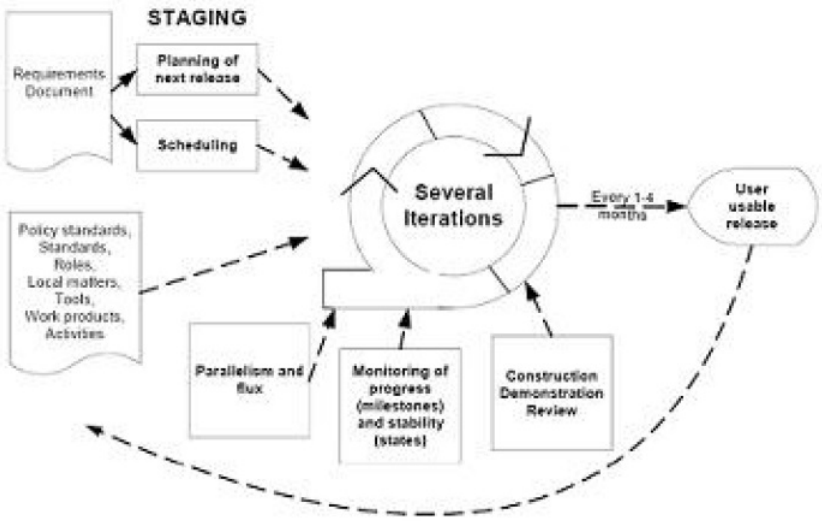

Figura 3.3: Fases do *Crystal*

- **Inspeção dos utilizadores**: São sugeridas duas a três inspeções realizadas por utilizadores a cada iteração;
- ***Workshops***: Reuniões que ocorrem antes e depois de cada iteração, com a finalidade de analisar os progressoas do projeto.

3.3 *Dynamic Systems Development Method (DSDM)*

Trata-se de uma metodologia Ágil de apoio ao desenvolvimento *software* e tem como foco principal o desenvolvimento de uma aplicação com a qualidade desejada sem ultrapassar os limites tanto de orçamento como de tempo.
Posto isto, o método DSDM tem em consideração a interação com o cliente e o utilizador final, realizando frequentemente protótipos por equipas autónomas, assim como teste durante todo o processo e uma lista de requisitos de forma priorizada. Esta ferramenta fornece uma *framework* para uma abordagem iterativa e incremental no desenvolvimento de um sistema.

3.3.1 Surgimento do conceito

A *Dynamic Systems Development Method (DSDM)* começou a ser desenvolvida nos anos 90 na Inglaterra onde foi aplicada como uma extensão RAD (*Rapid Application Development*) e é focada em projetos com prazos e orçamentos apertados. A DSDM aborda problemas que frequentemente ocorrem no desenvolvimento de informação que se prendem essencialmente com a falta de tempo, orçamentos apertados ou outro tipo de razões onde o projeto falhe, como por exemplo a falta de envolvimento dos encarregados do projeto ou dos utilizadores finais.

3.3.2 Conceitos

Na DSDM existem alguns conceitos importantes, nos quais esta metodologia assenta. Surgem deste modo os seguintes conceitos:

- ***Timeboxing*** - Técnica utilizada em projetos baseados na metodologia DSDM.

  A principal ideia por de trás deste conceito é dividir o projeto em pequenas porções, selecionando um número de requisitos de acordo com o princípio de *MoSCoW*, onde cada uma contém um orçamento fixo e uma data de entrega estipulada.

  Posto isto, se um dado projeto estiver a ficar sem orçamento ou tempo para ser desenvolvido, os requisitos com menor prioridade são omitidos. Isto significa que um produto incompleto pode ser entregue, devido aos princípios subjacentes à DSDM, onde 80% do projeto pode ser realizado em 20% do tempo que leva a construir o produto finalizado e de que nenhum sistema é construido sem falhas na primeira tentativa.

- ***MoSCoW*** - Técnica que representa um método de definição de prioridades nas tarefas efetuadas. Assim a técnica *MoSCoW* da DSDM é utilizada para prioritizar os requisitos enunciados.

  Este acrónimo representa *"MUST have this SHOULD have this if at all possible COULD have this if it does not affect anything else WON'T have this time but WOULD like in the future"*.

- **Prototipagem** - Técnica referente à criação do protótipo do sistema em desenvolvimento numa fase inicial do projeto. Possibilitando assim a descoberta antecipada de possíveis problemas no sistema e o teste por parte dos futuros utilizadores do produto.

  Assim, é conseguido um bom envolvimento do utilizador final com o sistema, sendo este um dos principais fatores de sucesso da DSDM.

- ***Workshop*** - Técnica centralizada em juntar os diferentes *stakeholders* de modo a fazer com que estes discutam os diferentes requisitos estipulados, assim como as funcionalidades do produto, levando a um entendimento mútuo.

- **Testes** - Técnica usada para garantir a qualidade de um produto. A DSDM obriga à realização de testes em todas a iterações realizadas. Uma vez que a DSDM é uma metodologia independente de ferramentas e técnicas, a quipa do projeto é livre de escolher o seu próprio método de teste.

3.3.3 Fases do DSDM

A metodologia *Dynamic Systems Development Method* consiste em três fases sequenciais: Pré-Projeto, Projeto e Pós-projeto [15]. Como é observável na figura 3.4

Pré-projeto (Fase 1) Fase onde o projeto é descrito e identificado, tratando-se posteriormente do seu plano financeiro e é assegurado um compromisso de realização. O facto destas questões serem tratadas numa fase inicial evita problemas futuros em fases mais avançadas do desenvolvimento do projeto.

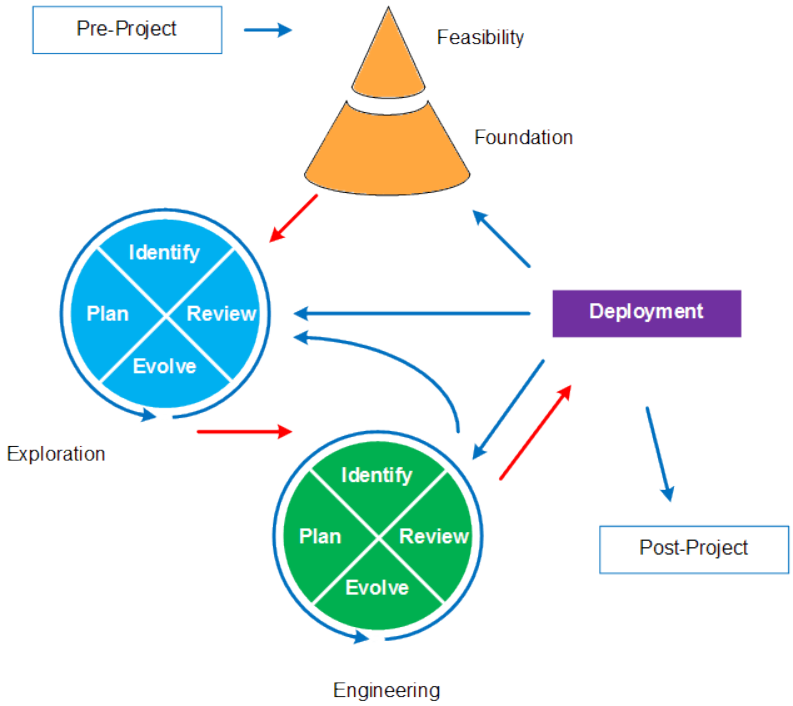

Figura 3.4: Metodologia DSDM

Projeto (Fase 2) A fase de Projeto do DSDM é a fase mais elaborada das três, baseando-se em 5 níveis formados por uma abordagem passo-a-passo e iterativa no desenvolvimento de um Sistema de Informação. Os dois primeiros níveis (Estudo de Viabilidade e Estudo de Negócio) são fases sequenciais que se complementam. Em seguida destas duas fases estarem completas, o sistema é desenvolvido iterativamente e de forma incremental nos níveis de Análise Funcional, Desenho e, por fim, Implementação.

Pós-projeto (Fase 3) Finalmente, a fase de Pós-projeto assegura um sistema de atuação eficiente, este é implementado através da manutenção e melhoramentos de acordo com os princípios da DSDM. É também possível a iniciação de novos projetos, com intuito de atualizar o sistema existente ou desenvolver um novo sistema.

3.4 *Extreme Programming (XP)*

O *Extreme Programming* trata-se de uma metodologia ágil para equipas pequenas e médias que desenvolvem *software* com requisitos pouco específicos que estão constantemente a ser alterados. Assim é adotada um estratégia de constante acompanhamento e de pequenas alterações ao longo do desenvolvimento do *software*, estas alterações são realizadas em ciclos idênticos ao descrito na figura 3.5.

3.4.1 Surgimento do conceito

A metodoligia XP (*Extreme Programming*) foi originada por *Kent Beck* durante o seu trabalho no projeto *Chrysler Comprehensive Compensation System (C3)* [19].

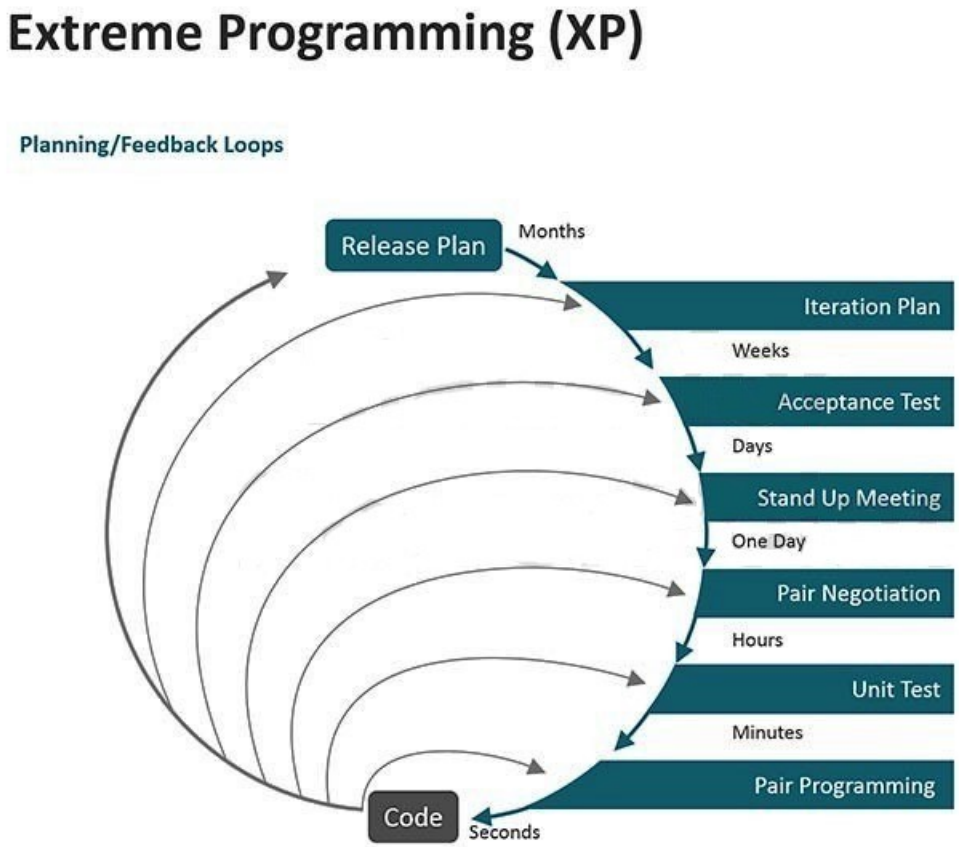

Figura 3.5: Metodologia XP

Assim, *Beck* tornou-se o líder do projeto em março de 1996, começando a refinar a metodologia de desenvolvimento usada no projeto

Posteriormente, *Kent Beck* foi convidade para o projeto C3 de modo a aumentar a performance do sistema, porém o seu cargo aumentou à medida que foi reparando em alguns problemas existentes no processo de desenvolvimento. *Beck* aproveitou a oportunidade para propor e implementar algumas alterações nas práticas originais do C3, que eram baseadas no seu trabalho com o seu colaborador *Ward Cunningham.*

Para além disto, *Ron Jeffries* também foi convidado para integrar o projeto com intuito de desenvolver e refinar estes métodos alterados. Acabando por atuar como treinador para incutir as práticas e hábitos na equipa do C3.

Posto isto, os princípios e práticas base do XP foram alterados por diferentes elementos através de discussões na wiki original, *Cunningham's WikiWikiWeb*, o que levou à criação de novas metodologias.

3.4.2 Valores

O XP assenta em cinco valores fundamentais, sendo eles: comunicação, simplicidade, *feedback*, coragem e respeito [7].

- **Comunicação** : Fator determinante para o sucesso de um projeto, devido

à interação entre os membros da equipa (programadores, clientes, treinador). Para desenvolver um produto a equipa precisa ter qualidade nos canais de comunicação estabelecidos. As discussões pessoalmente são sempre a melhor opção do que os outros canis existentes.

- ***Feedback*** : De modo a que o projeto tenha uma evolução positiva as respostas às decisões tomadas devem ser rápidas e visíveis. Deste modo, os elementos integrantes da equipa devem ter, em qualquer momento do projeto, a consciência do que está a acontecer.
- **Coragem** : Neste contexto, coragem está associada à iniciativa de alterar um código em produção, sem causar *bugs*, com agilidade. Esta tarefa é algo que exige algum grau de coragem e responsabilidade.
- **Simplicidade** : Para atender de forma rápida às necessidades do cliente, na generalidade dos casos um dos valores mais importantes é a simplicidade. Normalmente a expectativa do cliente é muito mais simples do que os programadores desenvolvem.
- **Respeito** : Numa equipa todos têm a sua importância e devem ser respeitados e valorizados.

Estes valores tem como princípios básicos o seguinte:

- *Feedback* rápido;
- Presumir a simplicidade;
- Mudanças incrementais;
- Aceitar mudanças;
- Trabalho de qualidade.

3.4.3 Variàveis de Controlo

A metodologia *Extreme Programming* baseia-se em algumas variáveis de controlo, variáveis estas que são geralmente usadas como variáveis de controlo de projetos (custo, tempo, qualidade e escopo), havendo um foco explícito no escopo. Deste modo é recomendada a priorização de funcionalidades que representam um valor mais elevado para o negócio. Posto isto, caso seja necessário diminuir o escopo, as funcionalidades menos valiosas serão adiadas ou canceladas.

O XP também incentiva ao controlo da qualidade como variável do projeto. Isto acontece devido ao facto de um pequeno ganho num curto prazo na produtividade, ao diminuir a qualidade, não é compensado por perdas a médio e longo prazo.

3.4.4 Papeis Adotados

- **Programadores** : Concentra-se na metodologia, sem hierarquia.
- **Treinador** : Pessoa com mais experiência que a equipa tem. É responsável por relembrar os restantes elementos das práticas e valores do CP. Este não necessita de ser o melhor programador mas sim o que entende e melhor aplica a metodologia.

- **Acompanhador** : Responsável por reunir para a equipa todos os dados, gráficos, informações que mostrem o andamento do projeto e ajudem a equipa a tomar decisões de implementação, arquitetura e design. Por vezes o próprio *coach* (treinador) faz o papel de *tracker* (acompanhador), outras vezes a equipa escolhe quem exerce este papel.
- **Cliente** : O cliente em XP é parte integrante da equipa. Este deve estar presente e pronto para responder a dúvidas dos programadores.

3.4.5 Práticas Adotadas

Esta metodologia sugere 12 práticas para o desenvolvimento de *software*. Estas práticas podem ser subdivididas em 4 grupos, [1], organizados de acordo com os valores e princípios definidos.
Práticas definidas:

- ***Feedback***
  - *Test-Driven Development*;
  - *Planning Game*;
  - *On-site Customer*;
  - *Pair Programming*.
- ***Continual Process*:**
  - *Code Refactoring*;
  - *Continuous Integration*;
  - *Small Releases*.
- ***Code understanding*:**
  - *Simple Design*;
  - *Coding Standards*;
  - *Collective Code Ownership*;
  - *System Metaphor*.
- ***Programmer's work conditions*:**
  - *40-Hour Week*.

3.5 *Scrum*

O *Scrum* assenta os seus princípios fundamentalmente em boas práticas de gestão, assumindo-se como uma metodologia ágil e flexível. Tem como objetivo definir um projeto iterativo e incremental de desenvolvimento de produtos ou gestão de projetos. Deste modo, produz um conjunto de funcionalidades que são cada vez mais próximas do objetivo final à medida que se termina cada iteração.
Este método é centralizado no trabalho em equipa, melhora a comunicação e maximiza a cooperação permitindo que cada elemento dê o seu melhor, aumentando assim a produtividade. O *Scrum* pode ser aplicado a projetos grandes e pequenos , esforçando-se para libertar o processo de quaisquer barreiras. O seu objetivo principal é conseguir uma avaliação correta do ambiente que está a ser desenvolvido. Para além disto, o *Scrum*, embora englobe processos de engenharia, não requer nem fornece qualquer técnica ou método específico para o desenvolvimento do *software*, apenas estabelece um conjunto de regras e práticas de gestão.

3.5.1 Surgimento do conceito

O *Scrum* foi desenvolvido por *Ken Schwaber* e *Jeff Sutherland* e inspirada nas ideias de desenvolvimento rápido e concorrente de produtos de *Hirotaka takeuchi* e *Ikujito*

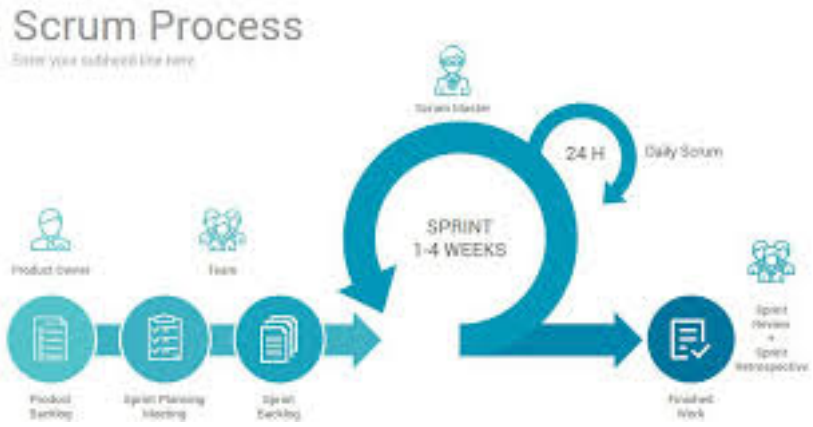

Figura 3.6: Metodologia Scrum

Nonaka. Assim nasceu a necessidade de encontrar uma metodologia que abordasse o problema do desenvolvimeto de *software* de uma forma não tradicional, que que a equipa age como um todo para atingir os seus objetivos.

3.5.2 Vocabulário Adotado

No Scrum é utilizada algum vocabulário específico[2].

- ***Backlog*** : Lista de todas as funcionalidades a serem desenvolvidas durante o projeto. Esta lista é definida no inicio do trabalho e deve ser descrita e orientada por prioridade de execução.
- ***Sprint*** : Período não superior a 30 dias no qual o projeto é desenvolvido (ou algumas funcionalidades).
- ***Sprint Backlog*** : Trabalho desenvolvido num sprint com intuito de criar um produto a apresentar ao cliente. Estas funcionalidades devem ser desenvolvidas de forma incremental, relativa ao *Backlog* anterior.
- ***Scrum Meeting*** : Reuniões realizadas diariamente onde são avaliados os progressos do projeto, assim como as barreiras encontradas durante o desenvolvimento.
- ***Scrum Rules*** : Protocolos que devem ser seguidos de modo a realizar uma reunião *Scrum*.
- ***Scrum Team*** : Equipa de desenvolvimento de um *sprint*.
- ***Scrum Master*** : Elemento que faz parte da equipa responsável pela gestão do projeto e liderar os *Scrum Meetings*.

3.5.3 Regras

No *Scrum* existe a necessidade de respeitar algumas regras de execução, mais especificamente no que diz respeito aos *Backlog*, *sprint* e *Scrum Meetings*.

Backlog: Tarefa de debate, em equipa, dos pontos que devem integrar a lista de funcionalidades da aplicação. Sendo que a responsabilidade de ordenar a lista por prioridade de execução é do *Scrum Master*.

Sprint: Deve ser realizado num período não superior a 30 dias, assim como não deve ter uma equipa que exceda os 9 elementos. Além disto, deve ter um objetivo bem claro aos olhos de todos os membros da equipa, baseado no *Backlog*, que não devem ser alterados enquanto o *sprint* está a decorrer. Porém, se aos olhos do *Scrum Master* houver a necessidade de alterar alguma funcionalidade, por motivos de ter influência no decorrer do projeto e que possam ser completadas no *sprint*, é possível realizar essa alteração.
Caso o *sprint* esteja a levar um rumo não desejável é possivel dissolver o *sprint* e começar um novo, baseando no *sprint Backlog*.

Scrum Meetings: Ponto de grande importância no desenvolvimento de um projeto. São realizadas reuniões onde o *Scrum Master* se atualiza do decorrer do projeto e procura identificar os pontos de conflito do desenvolvimento, podendo assim agir para os eliminar.
As reuniões devem ser realizadas diariamente, tentando que sejam realizadas dentro dos mesmo horários e no mesmo local, com uma duração a rondar os 30 minutos, no máximo. O diálogo deve ser guiado pelo *Scrum Master*, sendo restrito pelas perguntas impostas por ele, sendo elas:

1. O que foi desenvolvido desde a última reunião?
2. Que dificuldades foram encontradas durante o desenvolvimento?
3. O que está planeado para ser desenvolvido até à próxima reunião?

Todos os elementos, que têm atribuídos requisitos, devem responder às perguntas anteriores e, com base nas respostas obtidas, o *Scrum Master* deve tomar as decisões de forma a remover todas as possíveis situações que impeçam o funcionamento correto no desenvolvimento do projeto.

3.5.4 Processo

Inicialmente, para realizar o processo de *scrum*, o primeiro passo é decidir a constituição da equipa de trabalho. Esta equipa não deve ter mais do que 6 a 9 elementos. Se houverem mais membros do que é possível ao *Scrum Master* gerir estes são separados em várias equipas de *Scrum* onde cada uma focar-se-á em áreas específicas do projeto[2].
De seguida, deve ser decidido o *Scrum Master*, uma vez que este condus os *Scrum Meetings*, medindo o progresso e tomando as decisões necessárias para remover os obstáculos encontrados.
Finalmente, é vital para o sucesso do processo cumprir com os trabalhos rigorosamente, baseando nos restantes pontos do *Sprint Backlog*. Para isso é preciso estabelecer e conduzir as reuniões diárias, onde as equipas encontram e atualizam o estado atual do projeto. Assim, os elementos integrantes nas equipas tenham um foco diários no trabalho em desenvolvimento.

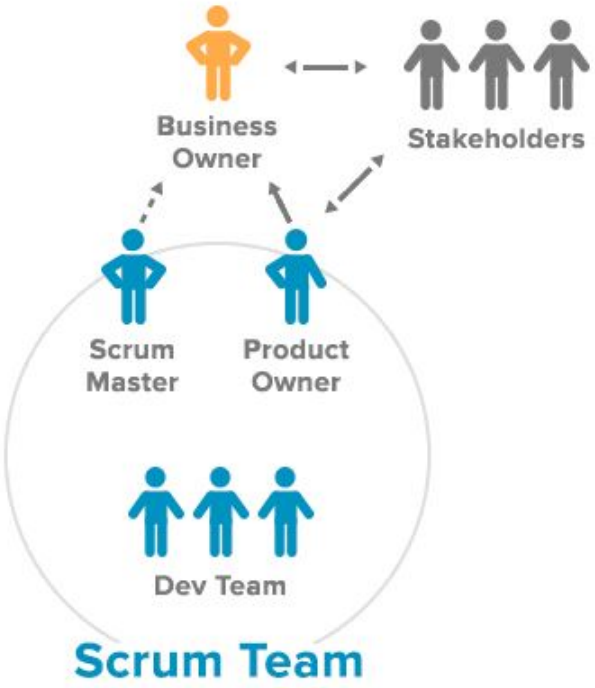

Figura 3.7: Elementos interveniente no projeto.

3.6 *Test Driven Development*

O *Test Driven Development* (TDD) é um método Ágil de desenvolvimento de *software* que relaciona os conceitos de verificação e validação, com um ciclo de repetições curto. Foi criado por *Kent Bec* em 2003, com os princípios que esta técnica originasse e encorajasse designs de código simples e que inpirassem confiança.

Inicialmente o programador tem que escrever um caso de teste automatizado que define uma melhoria desejada ou uma nova funcionalidade[16]. Em seguida, é produzido o código que pode ser validade pelo teste, código este que posteriormente é refatorado para um que contenha padrões aceitáveis.

3.6.1 Fases

O TDD é constituído for várias fases distintas[21]:

1. **Adicionar um Teste (Fase 1):** Cada nova funcionalidade é inicializada quando um teste é inserido. Cada teste novo tem grande probabilidade de falhar devido ao facto de ser escrito antes da funcionalidade ser implementada.

   Para escrever um teste o programador precisa de entender as especificações e requisitos do produto. Para que isto aconteça são necessários alguns User Stories que descrevam os requisitos e características mais especificas, tornando assim o programador mais focado nos requisitos antes de implementar ou bater código.

2. **Executar os Testes (Fase 2):** Validação se todos os testes que foram desenvolvidos estão a funcionar corretamente. Para além disso, inspeciona se

o novo requisito traz algum tipo de equívoco, continuando a não ser necessário nenhum código adicional.

Quanto ao novo teste, na generalidade falham devido ao facto de a funcionalidade ainda não ter sido desenvolvida, aumentando assim a confiança que se está a testar a coisa correto e que o teste apenas irá passar nos casos específicos que deve passar.

3. **Escrever Código (Fase 3):** Escrever código que irá fazer com que os testes anteriormente desenvolvidos passem. O novo código pode não ser perfeito, isto é, pode passar no teste mas estar mais "martelado"do que era desejável.

   Este aspeto é aceitável porque existem fases posteriores que se focam em melhorar/refatorar o código existente.

4. **Executar Testes (Fase 4):** Se os testes criados passarem todos com sucesso, o programador pode confiar no seu código e assumir que este possui todos os requesitos pretendidos.

5. **Refatorizar Código (Fase 5):** Fase onde o código é "limpo"e reorganizado, tornando-se assim mais percetível. Ao executar novamente os testes o programador deve constatar que as alterações realizadas não alteraram qualquer funcionalidade existente.

   Um dos passos mais frequentes de refatorização é eliminar código repetido, sendo considerando um aspeto importante no design de um *software*.

6. **Repetir (Fase 6):** Criação de um ou mais testes, o ciclo é então repetido. A evolução do projeto deve ser realizada de forma lente, se o novo código não satisfaz rapidamente um novo teste, ou outros testes falham inesperadamente, o autor do teste deve desfazer ou rever as alterações realizadas. O facto de realizarmos alterações pequenas continuamente ajuda na criação de pontos facilmente reversíveis.

Figura 3.8: Fases TDD.

3.6.2 Limitações

Foram também levantadas as seguintes limitações[21]:

- **Facto 1:** O desenvolvimento dirigido com testes é difícil de usar em situações onde os testes totalmente funcionais são requisitos para determinar o sucesso ou falha.

  O TDD encoraja os programadores a incluir o mínimo de código funcional em módulos e maximizar a lógica, usando *Fakes mocks* para representar o mundo externo.

- **Facto 2:** O suporte genérico é muito importante. O tempo gasto em testes é um desperdício se não for ponto acente que o TDD serve para melhorar o produto.

- **Facto 3:** Os testes são parte importante para realizar a manutenção do projeto. Existe diferentes formas de desenvolver testes, e testes mal escritos, suscetíveis a falhas por exemplo, são caros de manter. Surge sempre o risco em que os testes geram falsas falhas, testes estes que são ignorados, assim quando um teste real ocorre pode não ser detetada pelo programador.

  É fácil escrever testes de baixa e fácil manutenção, podendo ser este o objetivo durante a fase refatoração descrita anteriormente.

- **Facto 4:** Ainda nos testes, o nível de cobertura e detalhe alcançado durante os respetivos ciclos de TDD não pode ser facilmente re-criado numa data tardia. A existência de uma arquitetura pobre, mau *design* ou uma estratégia mal feita leva a uma mudança tardia, o que acaba por fazer que muitos testes falhem.

- **Facto 5:** Lacunas originadas na cobertura de testes inesperadamente. Pode acontecer que alguns elementos integrantes da equipa não usem o TDD, o que leva À escrita de forma errada dos testes, podendo mesmo dar origem a muitos conjuntos de testes inválidos, excluídos ou até desativados acidentalmente ou com intuito de serem melhorados posteriormente. Este fator pode levar a que haja uma quantidade elevada de testes a serem reescritos posteriormente fazendo com que serão corrigidos já tarde e a sua refatoração é mal feita.

- **Facto 6:** O facto de serem desenvolvidos um elevado número de testes pode levar a que haja uma falsa confiança na segurança no produto. Assim, vai haver um menor nível de atividade da garantia de qualidade, como testes de integração e aceitação.

- **Facto 7:** Cada teste é criado por um programador num ambiente de desenvolvimento especifico. Posteriormente, o mesmo programador irá escrever o código de modo a verificar a funcionalidade pretendida e que passe no deste. Assim os testes podem ter os mesmo "pontos cegos"que o código desenvolvido.

3.7 *Kanban*

A definição de *Kanban* nasceu da língua japonesa, que tem como significado "Cartão"ou "sinalização". Esta definição está relacionada com a utilização de cartões para indi-

car o andamento das tarefas alocadas ao longo do projeto, em cada cartão é colocada informação sobe uma determinada tarefa.
À medida que são colocados os cartões, os que já estavam lá colocado vão sendo organizados em listas que representam o processo de desenvolvimento do produto. Movendo-os também entre as diferentes secções à medida do desenvolvimento do projeto.

3.7.1 Surgimento do conceito

Em 1960 a*Toyota*[20] começou otimizar os seus processos baseando-se no mesmo modelo que os supermercados utilizavam para colocar o *stock* nas montras.
Deste modo, quando a *Toyota* aplicou o sistema tinha como objetivo alinhar melhor os seus níveis de *stock* com o consumo real. Para comunicar os níveis de capacidade, em tempo real, na fabrica os trabalhadores passavam um cartão entre as diferentes equipas. Quando os materiais necessários acabavam um *Kanban* era enviado para o depósito, onde era descrita a quantidade e o material necessário para o término das tarefas em linha. Por consequência, o depósito enchia um novo contentor com o material pedido que era posteriormente enviado.
Com o passar dos anos a tecnologia de sinalização do processo evoluiu, porém o processo de fabricação "*just in time*"(ou JIT)[20] continua a ser utilizado.

3.7.2 Vocabulário Adotado

- ***Just in Time (JIT)*** : Sistema de administração da produçao que determina tudo o que deve ser produzido, transportado ou comprado.
- ***Kanban* de Produção** : Cartão que autoriza a produção de um determinado objeto.
- ***Kanban* de Movimento** : Cartão que autoriza a movimentação física de peças entre o processo do fornecedor e o processo do cliente.

3.7.3 Processo

De modo a colocar em prática a metodologia *Kanban* deve seguir-se os seguintes passos[21]:

1. Definir os estados possíveis da tarefa, como "Por Fazer", "Em Desenvolvimento", "Feito", entre outros.
2. Usar uma ferramenta de suporte à representação destes estados que irão formar as listas de tarefas. (Quadro, folha de cálculo, ferramenta de gestão de tarefas).
3. Listaras tarefas designadas a serem desenvolvidas num determinado período de tempo, tarefas estas que devem ser colocadas no grupo "Por Fazer".
4. Sempre que alguém começar uma determinada tarefa deve de imediato alterar o estado da tarefa para "Em Desenvolvimento"de forma a que os restantes membros da equipa saibam que alguém já começou a tarefa.

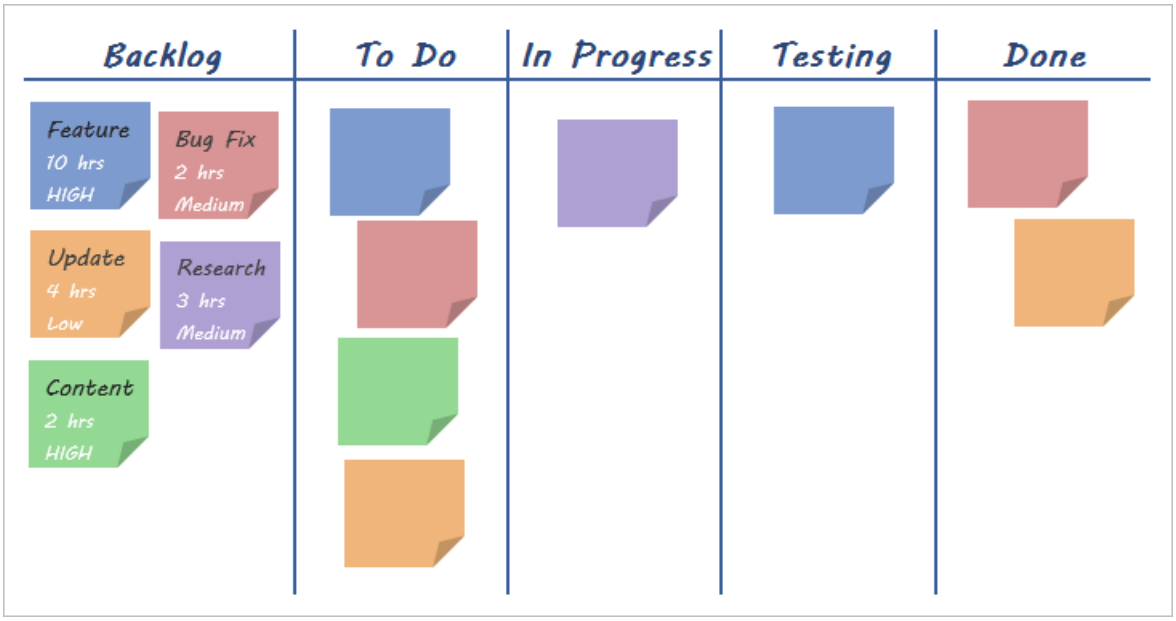

Figura 3.9: Metodologia *Kanban*

.

5. Quando uma dada tarefa é terminada, o responsável pela mesma deve alterar o "local"da tarefa para o estado seguinte no fluxo definido inicialmente.

   Em alguns casos, pode acontecer de definir a tarefa para ser sujeita a revisão ou ainda para ser alvo de testes unitários, o que significa que existe uma outra tarefa associada que consistiria em desenvolver os testes para a tarefa que foi terminada.

O *Kanban* é considerado um aliado ao *Scrum*, assim como a outras metodologias. Isto acontece uma vez que o *Kanban* por si só não define limites temporais para cada tarefa/ conjunto de tarefas, não existindo os chamados *Sprints*. Logo este funciona como auxilio ao bom funionamento destes *sprints*, dando garantias a que nada falte.

Capítulo 4

Diferenças das metodologias

Uma vez que já foram vistos os diferentes métodos Ágeis, mais usados nos dias de hoje, surge a necessidade de os comparar com a finalidade de retirar algumas conclusões e enquadrar cada um deles nos diferentes quadrantes, segundo as suas características.
Com a existência de tantas metodologias é normal haver diferenças nas abordagens utilizadas para os projetos realizados, é também natural não usar a mesma abordagem numa equipa de seis elementos, do que uma equipa onde existem sessenta ou até mesmo cem pessoas. Para diferentes situações situações existem abordagens diferentes, sendo necessário alinhar a abordagem mediante as variáveis do projeto em desenvolvimento[13], assim é necessário dividir os diferentes métodos em diferentes categorias.

4.1 Necessidades

Como existem vários métodos para desenvolver um *software*, para haver sucesso no projeto em desenvolvimento é importante haver uma certa flexibilidade na escolha do método a ser utilizado[4]. Isto é importante pelas seguintes razões:

- **Diferentes tecnologia requerem diferentes técnicas:** Existem diferentes metodologias que se enquadram melhor ou pior dependendo da finalidade do projeto que se está a desenvolver.
- **Cada pessoa é única:** Cada elemento da equipa tem um *background* diferente, personalidade diferente, preferências e gostos. Assim o método que resulta bem numa pessoa pode não resultar bem com outra.
- **Cada equipa é única:** As equipas contem elementos, assim sendo, como as pessoas são únicas vão formar também equipas únicas. Sendo um ponto muito importante adaptar o método de trabalho à equipa.
- **As necessidades exteriores podem variar:** É importante adaptar o método de trabalho de acordo com o meio em que o projeto se enquadra, isto para incorporar os fatores externos existentes no projeto.
- **As categorias dos projeto variam:** Cada projeto requer a sua abordagem especifica. Cada um tem a sua categoria e as suas prioridades e objetivos.

- **As categorias dos métodos variam:** Os métodos ágeis são diferentes, contendo cada um as suas vantagens e desvantagens.

4.2 Categorias dos métodos

Existem quatro categorias nas quais os métodos de desenvolvimento podem ser separados, onde cada categoria apela a uma mentalidade diferente.

4.2.1 Definição das Categorias

As diferentes categorias definidas foram:

- **Code and Fix -**[3] Abordagem também conhecida como"*hacking*". Por norma, é uma abordagem onde não existe planeamento, e quando existe o plano é rapidamente abandonado. Os cronogramas e estimativas realizados raramente sãi sumpridos.
- **Iterativo Rigoroso -**[10] Processos bem definifos que incluem procedimentos detalhados onde os programadores aplicas os mesmos de maneira iterativa.

  Neste caso, os requisitos do projeto são definidos inicialmente em alto nível, com alguns detalhes onde posteriormente são detalhados conforme o necessário. Estes projetos também vão sendo entregues numa base incremental após curtos ciclos de lançamento.
- **Àgil -**[14] Abordagem orientada a pessoas, permite que respondam efetivamente às mudanças e resulta na criação de sistemas de trabalho que entendam às necessidades dos *stackholders*. Os diferentes processos são definidos em alto nível, geralmente apresentados como uma coleção de práticas ou filosofias.

4.2.2 Finalidade das Categorias

Como foi referido anteriormente, existem diferentes categorias, com diferentes finalidades, para englobar os métodos ágeis que são utilizados.Posto isto, cada categoria tem as seguintes finalidades:

- **Code and Fix -** Métodos apropriados para programadores independentes.
- **Iterativo Rigoroso -** Apropriados para equipas que pretendem ter uma abordagem inicial simples. São utilizados geralmente em indústria muito regulamentadas e burocráticas.
- **Àgil -** Abordagem nova e que normalmente permite ter em conta todos os *stakeholders* do projeto.

4.3 Comparação ente os métodos

Existem gráficos que relacionam os vários métodos, figura 4.1, tendo em conta a adaptação a cada projeto, sendo este o âmbito deste ensaio. Para além disso, também é relacionado o tamanho do ciclo do projeto ou se será necessário adotar um outro métodos posteriormente.

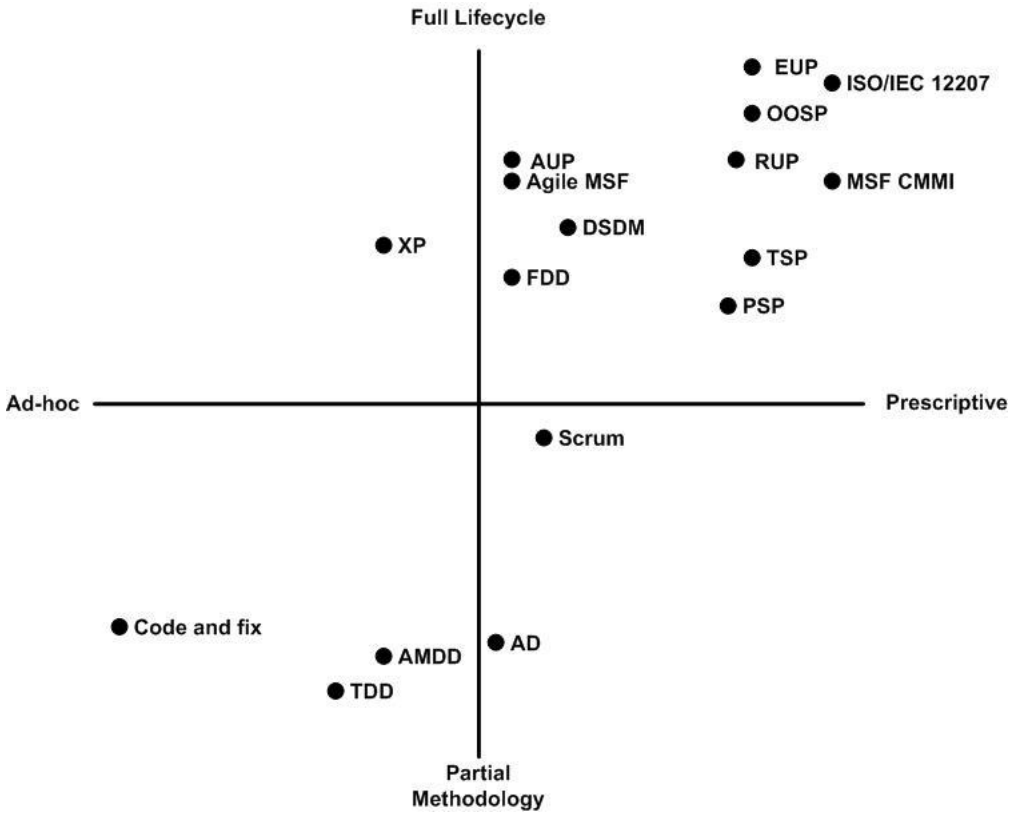

Figura 4.1: Comparação entre os diferentes métodos.

.

4.4 Necessidade do uso de diferentes métodos

Para realizar a comparação entre os métodos foram "pesados"os prós e contras dos métodos a cima descritos. Estes métodos foram selecionados pelo facto de nem todas as empresas estarem pré-dispostas ou não tem as características necessárias para adotar uma metodologia Ágil na sua organização interna. Como já foi referido anteriormente, existem alguns métodos Ágil que não servem tudo, nem são apropriados para tudo. Assim é necessário ter uma mente aberta e a capacidade de escolher o melhor método para a altura adequada.

Na mesma organização é possível adotar vários métodos dos mencionados, desde que se tenha em conta o seguinte:

- Não haver variações na quantidade de métodos e aplicar apenas um pequeno conjunto que sirva a grande maioria dos projetos desenvolvidos.
- Realizar um esforço inicial para inicial de trabalhar e modificar o método para se moldar à equipa que o vai usar.
- Definir objetivos comuns a todos os projetos em vez de procedimentos, pois cada projeto é único.
- Definir de forma clara a terminologia utilizada.
- Flexibilidade.
- Priorizar corretamente e desde o inicio os requisitos e objetivos.

Capítulo 5

Diagnostico Ágil

Após o estudo dos diferentes métodos ágeis é possível identificar uma série de características em comum entre eles, características estas que permitem distinguir as metodologias Ágil das metodologias mais tradicionais[4]. Ver qual a metodologia mais adequada para adotar ao desenvolver um projeto depende de vários fatores, como por exemplo, número de recursos disponíveis, o tipo de projeto em que se vai trabalhar, o tempo que temos para o desenvolvimento do projeto, entre outros[11]. Posto isto, o diagnóstico ágil é uma técnica que permite ,à equipa responsável pelo desenvolvimento, verificar se adota uma das metodologias descritas anteriormente ou é melhor optar por uma metodologia mais tradicional.

É sabido que as metodologias ágeis, em comparação ás metodologias mais tradicionais, trás muitas vantagens, principalmente na indústria do software. Apesar deste fator, a metodologia Ágil não deve se utilizada em qualquer projeto, antes de partir para o desenvolvimento é necessário definir um planeamento completo de modo a perceber se os recursos disponíveis são suficientes.

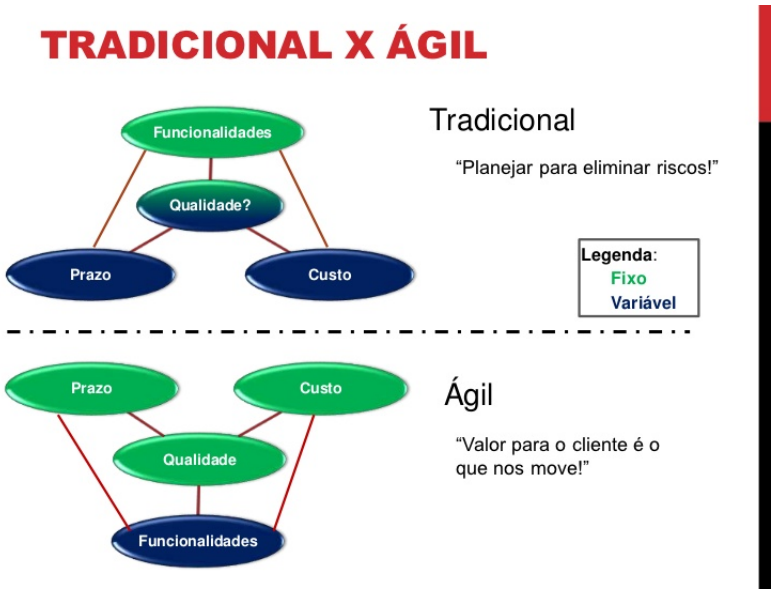

Figura 5.1: Comparação entre os métodos ágeis e os tradicionais.

.

5.1 Foco do trabalho

Posto isto, o foco deste trabalho propõe um questionário de forma a perceber se, uma dada equipa que vai elaborar um dado projeto, se deve basear em métodos ágeis ou optar por métodos tradicionais. O questionário contém algumas perguntas, focadas em equipas projeto diferentes, onde ,no fim, é indicado se um determinado projeto está a implementar corretamente a metodologia ou não.

Diagnóstico da Metodologia sobre um Projeto Ágil

Este diagnóstico deve tem como finalidade ser utilizado numa fase muito inicial do projeto, fase de pré-planeamento, e servirá como uma escolha entre os métodos mais tradicionais e os métodos ágeis. Como primeira versão, este diagnóstico apenas serve para indicar á equipa projeto se deve ou não utilizar uma metodologia ágil para realizar o projeto.

O questionário é constituído por 10 perguntas, onde é possível obter a cotação de 0,1 ou 2 de acordo com a resposta dada.

Questão 1: Dentro do grupo de projeto, sente a necessidade de alterar o método de desenvolvimento do projeto?

Sim(1 Pontos) **Não**(0 Ponto)

Questão 2: O projeto que vai ser realizado tem um curto prazo de entrega?

Sim(2 Pontos) **Não**(0 Ponto)

Questão 3: O cliente final necessita de documentação clara de cada etapa realizada?

Sim(1 Pontos) **Não**(2 Ponto)

Questão 4: O cliente final necessita de estar sempre a par e aprovar cada um dos diferentes requisitos desenvolvidos?

Sim(0 Pontos) **Não**(2 Ponto)

Questão 5: O projeto envolve múltiplas variantes?

Sim(2 Pontos) **Não**(1 Ponto)

Questão 6: O projeto está bem documentado e os requisitos estão todos definidos anteriormente?

Sim(0 Pontos) **Não**(2 Ponto)

Questão 7: A comunicação entre os elementos da sua equipa é boa?

Sim(1 Pontos) **Não**(0 Ponto)

Questão 8: Os elementos da sua equipa são pró-ativos e demonstram iniciativa?

Sim(1 Pontos) **Não**(0 Ponto)

Questão 9: Qual é a área do projeto?

Software ou Hardware(2 Pontos) **Serviços da Empresa**(1 Ponto) **Outros**(0 Ponto)

Questão 10: No seu ponto de vista, qual é a dimensão do projeto que vai realizar?

Pequeno(2 Pontos) **Médio**(1 Ponto) **Grande**(0 Ponto)

- **Menos de 6 pontos** - Os Métodos Ágil não são, à partida, apropriados ao seu projeto.
- **Entre 7 e 11 pontos** - Os Métodos Ágil talvez sejam apropriados mas necessita de uma segunda opinião.
- **Mais de 12 pontos** - Os Métodos Ágil são apropriados ao seu projeto.

Capítulo 6

Aplicação do Diagnóstico

Após a elaboração do diagnóstico este foi aplicado a três projetos distintos, projetos estes que contêm um número distinto de elementos nas equipas de desenvolvimento. O primeiro projeto é desenvolvido no contexto académico e os restantes tratam-se de projetos desenvolvidos em empresas. Os projetos de empresas têm duas áreas de foco distintas.

6.1 Projeto Académico

Projeto desenvolvido por alunos do Mestrado em Engenharia Informática na unidade curricular Engenharia Web. Grupos no máximo 3 elementos onde a entrega final tem o prazo até dia 30 de Maio, sendo que existe uma entrega intermédia no dia 16 de Maio, o que dá cerca de 3 meses para desenvolver o projeto.

Neste projeto é necessário desenvolver todo o backend e frontend da aplicação, onde existem cerca de 11 requisitos.

Diagnóstico da Metodologia sobre um Projeto Ágil

Este diagnóstico deve tem como finalidade ser utilizado numa fase muito inicial do projeto, fase de pré-planeamento, e servirá como uma escolha entre os métodos mais tradicionais e os métodos ágeis. Como primeira versão, este diagnóstico apenas serve para indicar á equipa projeto se deve ou não utilizar uma metodologia ágil para realizar o projeto.

O questionário é constituído por 10 perguntas, onde é possível obter a cotação de 0,1 ou 2 de acordo com a resposta dada.

Questão 1: Dentro do grupo de projeto, sente a necessidade de alterar o método de desenvolvimento do projeto?

Sim(1 Pontos) ***Não****(0 Ponto)*

Questão 2: O projeto que vai ser realizado tem um curto prazo de entrega?

Sim*(2 Pontos)* **Não**(0 Ponto)

Questão 3: O cliente final necessita de documentação clara de cada etapa realizada?

Sim(1 Pontos) ***Não****(2 Ponto)*

Questão 4: O cliente final necessita de estar sempre a par e aprovar cada um dos diferentes requisitos desenvolvidos?

Sim*(0 Pontos)* **Não**(2 Ponto)

Questão 5: O projeto envolve múltiplas variantes?

Sim(2 Pontos) ***Não****(1 Ponto)*

Questão 6: O projeto está bem documentado e os requisitos estão todos definidos anteriormente?

Sim*(0 Pontos)* **Não**(2 Ponto)

Questão 7: A comunicação entre os elementos da sua equipa é boa?

Sim*(1 Pontos)* **Não**(0 Ponto)

Questão 8: Os elementos da sua equipa são pró-ativos e demonstram iniciativa?

Sim*(1 Pontos)* **Não**(0 Ponto)

Questão 9: Qual é a área do projeto?

Software ou Hardware*(2 Pontos)* **Serviços da Empresa**(1 Ponto) **Outros**(0 Ponto)

Questão 10: No seu ponto de vista, qual é a dimensão do projeto que vai realizar?

Pequeno(2 Pontos) ***Médio****(1 Ponto)* **Grande**(0 Ponto)

O total de pontos obtido no diagnóstico é de **10 pontos**. Posto isto, o projeto encontra-se no nível intermédio, sendo aconselhada uma segunda opinião se é ou não aconselhado o uso de Métodos Ágeis.

6.2 Projeto Empresarial 1

Projeto de desenvolvimento no âmbito do *machine learning* por uma empresa. Este projeto deverá ser capaz de receber diversos estímulos e corresponder aos mesmos de diferentes formas. A equipa de desenvolvimento será constituída por 2 elementos e o prazo para o desenvolvimento do mesmo é até dia 13 de agosto, o que desde o inicio até á entrega final corresponde a um período de 3 meses. O cliente não definiu previamente um conjunto de requisitos, apenas alguns pedidos/reações que gostaria de ver implementados. Assim os requisitos vão sendo definidos à medida que o projeto for implementado.

Diagnóstico da Metodologia sobre um Projeto Ágil

Este diagnóstico deve tem como finalidade ser utilizado numa fase muito inicial do projeto, fase de pré-planeamento, e servirá como uma escolha entre os métodos mais tradicionais e os métodos ágeis. Como primeira versão, este diagnóstico apenas serve para indicar á equipa projeto se deve ou não utilizar uma metodologia ágil para realizar o projeto.

O questionário é constituído por 10 perguntas, onde é possível obter a cotação de 0,1 ou 2 de acordo com a resposta dada.

Questão 1: Dentro do grupo de projeto, sente a necessidade de alterar o método de desenvolvimento do projeto?

Sim(1 Pontos) ***Não****(0 Ponto)*

Questão 2: O projeto que vai ser realizado tem um curto prazo de entrega?

Sim*(2 Pontos)* **Não**(0 Ponto)

Questão 3: O cliente final necessita de documentação clara de cada etapa realizada?

Sim*(1 Pontos)* **Não**(2 Ponto)

Questão 4: O cliente final necessita de estar sempre a par e aprovar cada um dos diferentes requisitos desenvolvidos?

Sim*(0 Pontos)* **Não**(2 Ponto)

Questão 5: O projeto envolve múltiplas variantes?

Sim(2 Pontos) ***Não****(1 Ponto)*

Questão 6: O projeto está bem documentado e os requisitos estão todos definidos anteriormente?

Sim(0 Pontos) ***Não****(2 Ponto)*

Questão 7: A comunicação entre os elementos da sua equipa é boa?

Sim*(1 Pontos)* **Não**(0 Ponto)

Questão 8: Os elementos da sua equipa são pró-ativos e demonstram iniciativa?

Sim*(1 Pontos)* **Não**(0 Ponto)

Questão 9: Qual é a área do projeto?

Software ou Hardware*(2 Pontos)* **Serviços da Empresa**(1 Ponto) **Outros**(0 Ponto)

Questão 10: No seu ponto de vista, qual é a dimensão do projeto que vai realizar?

Pequeno*(2 Pontos)* **Médio**(1 Ponto) **Grande**(0 Ponto)

Após realizado o diagnóstico obteve-se um total de **12 pontos**. Assim concluí-se que o projeto em desenvolvimento se encontra no nível 3 definido anteriormente, sendo então aconselhado o uso de métodos ágeis para a realização do mesmo.

6.3 Projeto Empresarial 2

Projeto realizado no âmbito de uma empresa com a finalidade de desenvolvimento de uma aplicação web e mobile. Esta aplicação deverá ser capaz de receber diversos pedidos de reserva de mesas em restaurantes aderentes à plataforma.

A equipa de desenvolvimento é constituída por 10 elementos, distribuídos entre programadores, *designers* e direção. Este projeto tem um tempo estimado de 6 meses com alguns checkpoints a serem realizados durante o seu desenvolvimento.

Para além disto conjunto de requisitos já foram previamente definidos pelo cliente quando foi proposto o mesmo projeto.

Diagnóstico da Metodologia sobre um Projeto Ágil

Este diagnóstico deve tem como finalidade ser utilizado numa fase muito inicial do projeto, fase de pré-planeamento, e servirá como uma escolha entre os métodos mais tradicionais e os métodos ágeis. Como primeira versão, este diagnóstico apenas serve para indicar á equipa projeto se deve ou não utilizar uma metodologia ágil para realizar o projeto.

O questionário é constituído por 10 perguntas, onde é possível obter a cotação de 0,1 ou 2 de acordo com a resposta dada.

Questão 1: Dentro do grupo de projeto, sente a necessidade de alterar o método de desenvolvimento do projeto?

Sim(1 Pontos) ***Não****(0 Ponto)*

Questão 2: O projeto que vai ser realizado tem um curto prazo de entrega?

Sim(2 Pontos) ***Não****(0 Ponto)*

Questão 3: O cliente final necessita de documentação clara de cada etapa realizada?

Sim(1 Pontos) ***Não****(2 Ponto)*

Questão 4: O cliente final necessita de estar sempre a par e aprovar cada um dos diferentes requisitos desenvolvidos?

Sim(0 Pontos) ***Não****(2 Ponto)*

Questão 5: O projeto envolve múltiplas variantes?

Sim*(2 Pontos)* **Não**(1 Ponto)

Questão 6: O projeto está bem documentado e os requisitos estão todos definidos anteriormente?

Sim*(0 Pontos)* **Não**(2 Ponto)

Questão 7: A comunicação entre os elementos da sua equipa é boa?

Sim*(1 Pontos)* **Não**(0 Ponto)

Questão 8: Os elementos da sua equipa são pró-ativos e demonstram iniciativa?

Sim*(1 Pontos)* **Não**(0 Ponto)

Questão 9: Qual é a área do projeto?

Software ou Hardware*(2 Pontos)* **Serviços da Empresa**(1 Ponto) **Outros**(0 Ponto)

Questão 10: No seu ponto de vista, qual é a dimensão do projeto que vai realizar?

Pequeno(2 Pontos) ***Médio****(1 Ponto)* **Grande**(0 Ponto)

Após realizado o diagnóstico obteve-se um total de **11 pontos**. Assim concluí-se que o projeto em desenvolvimento encontra-se no nível intermédio, sendo aconselhada uma segunda opinião se é ou não aconselhado o uso de Métodos Ágeis.

Capítulo 7

Conclusão

Depois de descritas e analisadas as diferentes metodologias ágeis existentes, é de realçar as melhorias notórias que abrangem os processos e fluxos de trabalho das empresas, originando maior organização, capacidade de produção e consequentemente, satisfação dos clientes.
É de notar também que estas metodologias não são propriamente recentes, tendo só sido utilizadas e popularizadas na área da produção de Software há relativamente pouco tempo e o seu uso tem vindo a tornar-se cada vez mais comum. Apesar das vantagens das abordagens mais Ágeis, elas têm vantagens e desvantagens e não servem para todo o tipo de projeto.

Capítulo 8

Trabalho Futuro

O método proposto tem algumas características que geralmente são prioritárias: é simples, não demora muito a responder, é direto e ajuda a ter uma primeira perceção do método de trabalho a seguir.
É também importante definir que esta é apenas uma primeira versão do método de diagnóstico, será esperado que como trabalho futuro se sugira uma segunda parte do questionário que apenas seja respondida se a primeira parte indicar ser necessária uma segunda opinião para o uso da metodologia ágil, e onde se consiga perceber em específico se é adequado o uso destas metodologias no processo a ser inicializado ou não. Para além disso é também importante validar as questões num contexto real em projetos já desenvolvidos e onde se consiga ter uma perceção se as abordagens ágeis ou tradicionais resultaram.

Bibliografia

[1] alexSoft. Extreme programming: Values, principles, and practices. https://www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/. [Online].

[2] C. H. P. M. Bernardo Vasconcelos de Carvalho. Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica. http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-530X2012000300009. [Online].

[3] R. K. S. Carvalho. Aula 4- engenharia de software. https://pt.slideshare.net/RudsonKiyoshi/aula-4-engenharia-de-software. [Online].

[4] S. R. da Costa Ferraz. Recomendações para a adoção de práticas ágeis no desenvolvimento de software: estudo de casos. 2016.

[5] M. M. da Silva. Metodologia ágil das desenvolvimento adaptativo software. 2016.

[6] L. Gonçalves. Metodologia agile, tudo o que precisa de saber sobre este tema. https://luis-goncalves.com/pt-pt/o-que-e-metodologia-agile/. [Online].

[7] L. Lindstrom and R. Jeffries. Extreme programming and agile software development methodologies. *Information systems management*, 21(3):41–52, 2004.

[8] A. v. B. Mike Beedle. *Manifesto for Agile Software Development*. Agile Alliance.

[9] MindTools.com. Timeboxing maximizing your productivity. https://www.mindtools.com/pages/article/timeboxing.htm. [Online].

[10] D. Santos. Desenvolvimento iterativo e incremental. https://www.linkedin.com/pulse/desenvolvimento-iterativo-e-incremental-david-santos-csm-cmmi. [Online].

[11] D. T. Sato. Uso eficaz de métricas em métodos ágeis de desenvolvimento de software. *Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo*, 139, 2007.

[12] T. SIMPLUEASYLEARNING. Adaptive software development. https://www.tutorialspoint.com/adaptive_software_development/adaptive_software_development_tutorial.pdf. [Online].

[13] M. Soares. Comparação entre metodologias Ágeis e tradicionais para o desenvolvimento de software. *INFOCOMP*, 3(2):8–13, 2004.

[14] A. Tecnologia. O que configura um método de desenvolvimento ágil? `https://www.alwitecnologia.com.br/post/4/o-que-configura-um-metodo-de-desenvolvimento-agil`. [Online].

[15] D. Teixeira, F. Jorge, A. Pires, T. Alexandre, and G. Pereira Santos. Dsdm - dynamic systems development methodology. 01 2005.

[16] A. M. Tiryaki, S. Öztuna, O. Dikenelli, and R. C. Erdur. Sunit: A unit testing framework for test driven development of multi-agent systems. In *International Workshop on Agent-Oriented Software Engineering*, pages 156–173. Springer, 2006.

[17] L. R. R. Vanessa Finoto. Metodologia Ágil: Família crystal de cockbum. `https://pt.slideshare.net/vanessafinoto/metodologia-gil-famlia-crystal-de-cockbum`. [Online].

[18] Wikipedia. Crystal methods. `https://en.wikiversity.org/wiki/Crystal_Methods`. [Online].

[19] Wikipedia. Extreme programming. `https://en.wikipedia.org/wiki/Extreme_programming`. [Online].

[20] Wikipedia. Kanban. `https://pt.wikipedia.org/wiki/Kanban`. [Online].

[21] Wikipedia. Test driven development. `https://pt.wikipedia.org/wiki/Test_Driven_Development`. [Online].